HPC Challenge 2014
PCJ Benchmarks
(Parallel Computing in Java)

Marek Nowicki, Lukasz Górski, Piotr Bala

bala@icm.edu.pl

N. Copernicus University, Torun, Poland
ICM - University of Warsaw, Warsaw, Poland

Parallel computing in Java – challenges

- Parallel programming is still difficult especially while traditional programming paradigms are used
- There is need for new programing paradigms such as Partitioned Global Address Space (PGAS)
- HPC marked has to open for new languages widely used for data analysis such as Java
- Parallel programming in Java is either threads or fork/join and is limited to a single JVM
- There has been number of parallel extensions to Java however none of them become popular

PCJ - Parallel Computations in Java

Java library developed at ICM

- pcj.icm.edu.pl

Programming paradigm:

- partitioned global address space (PGAS)
- all variables are local by default
- variables can be global (@Shared)
- one sided communication (put, get)

Features

- does not require modification of JVM
- does not require other libraries!
- works on *almost all operating system that have JVM*
- uses newest Java SE 7 (NIO, SDP, . . .)

PCJ - Parallel Computations in Java

Basic functionality of PCJ:

- tasks numbering
- synchronization of tasks
- getting values
- putting values

Advanced functionality:

- broadcasting values
- monitoring variables
- parallel I/O
- creating groups of nodes
- working with groups.

PCJ - Hello world

```
import org.pcj.*
public class PcjHelloWorld extends Storage
                                            implements StartPoint {

    @Override
    public void main() {
          System.out.println("Hello!");
    }

    public static void main(String[] args) {
        String[] nodes = new String[]{"localhost", "localhost"};
        PCJ.deploy(PcjHelloWorld.class,
                            PcjHelloWorld.class, nodes);
    }
}
```

PCJ - basics

```
@Shared double a;
double c;

if (PCJ.myId()==0) c =(double) PCJ.get(3, "a");

FutureObject aL[] = new FutureObject[PCJ.threadCount()];
if (PCJ.myId()==0) aL[p] = PCJ.getFutureObject(p, "a");
c =(double) aL[p].get();

if (PCJ.myId()==0) PCJ.put(3, "a", 5.0);

public static void PCJ.barrier();
public static int PCJ.threadCount()
```

PCJ - Reduction

```
@Shared double a
FutureObject aL[] = new FutureObject[PCJ.threadCount()];
double a0 = 0.0;
 if (PCJ.myId() == 0) {
    for (int p = 0; p < PCJ.threadCount(); p++) {
        aL[p] = PCJ.getFutureObject(p, "a");
    }
    for (int p = 0; p < PCJ.threadCount(); p++) {
        a0 = a0 + (double) aL[p].get();
      }
   }
```

HPC Challenge PCJ benchmarks

HPC Benchmarks

STREAM	180 LOC
Random Access	146 LOC
GlobalFFT 1D	498 LOC

Our benchmarks

MapReduce	126 LOC
RayTracing	1627 LOC (incl. 100 comment lines) 52 PCJ calls, incl. 35 log statements

```
long sum = 0;
  for ( User user : users ) { um += user.getAge (); }
  double average = (double) sum / users.size ();
```

MapReduce - Java

- **Java**

```
long sum = 0;
for ( User user : users ) {
  um += user.getAge ();
}
double average = (double) sum / users.size ();
```

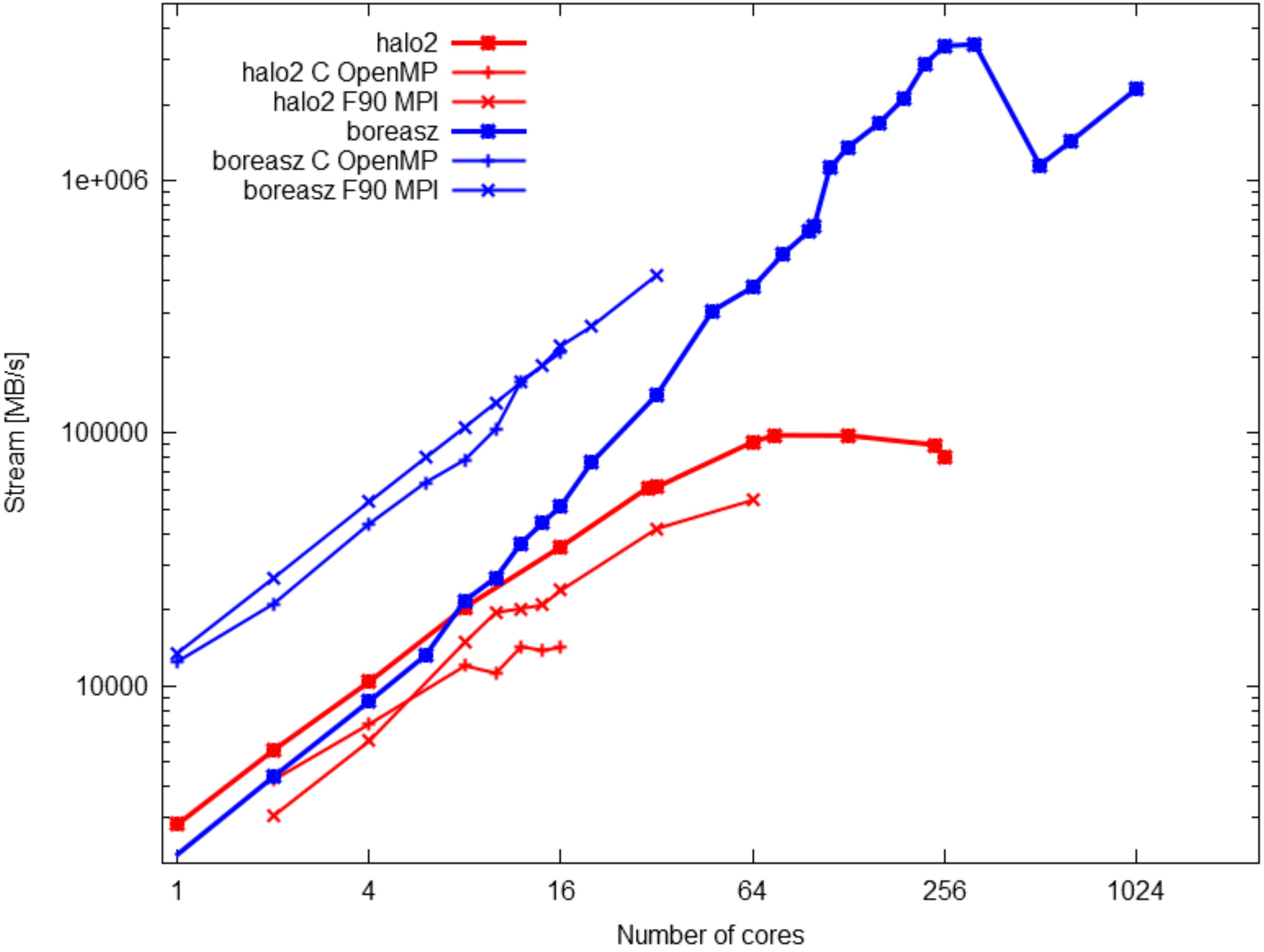
- **Java 8 parallel streams**

```
long sum = users.parallelStream ()
                    .map (u -> ( long ) u. getAge ())
                    .reduce ( Long :: sum )
                    .get ();
double average = (double) sum / users.size ();
```

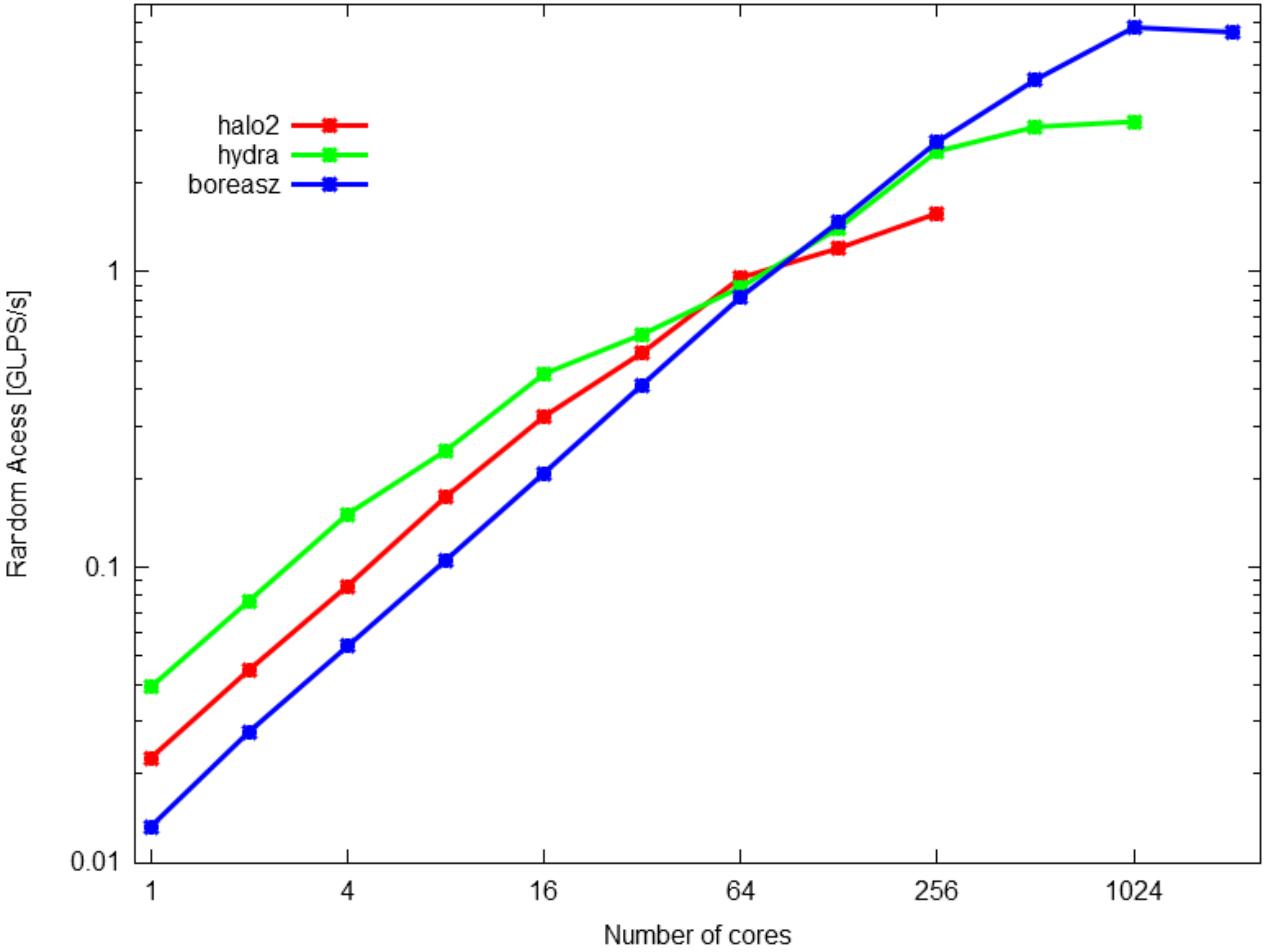
MapReduce - PCJ

```
@Shared  long sum ;
@Shared  int usersCount ;
...
myUsers = loadUsers( PCJ.myId ());
long s = 0;
for ( User u : myUsers ) {
        s += u. getAge ();
}
PCJ.putLocal ("sum", s);                         // The same for size
PCJ.barrier ();
s = pcj_reduce ("sum");
double average = (double) s / count ;
```

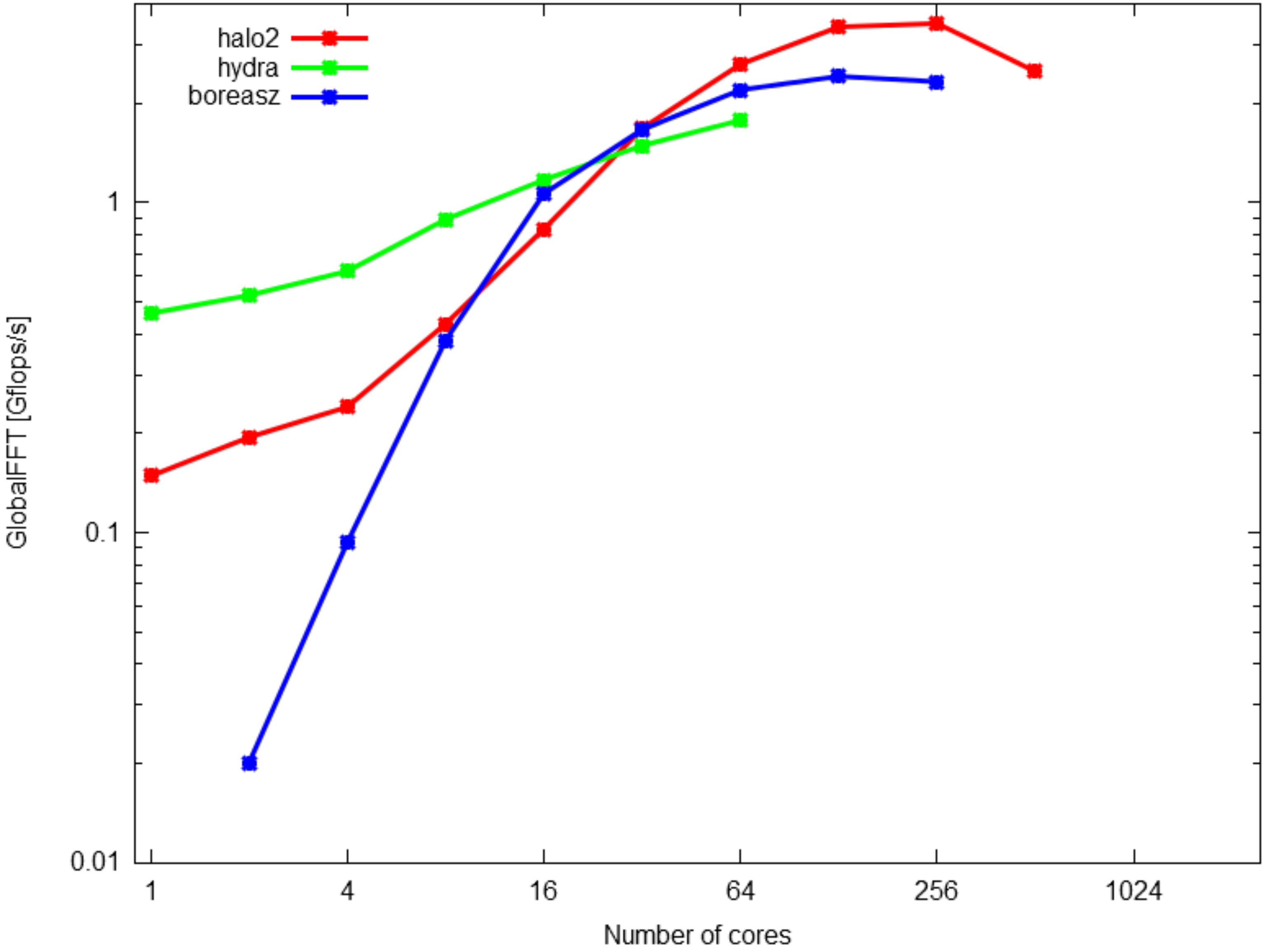
PCJ performance – STREAM



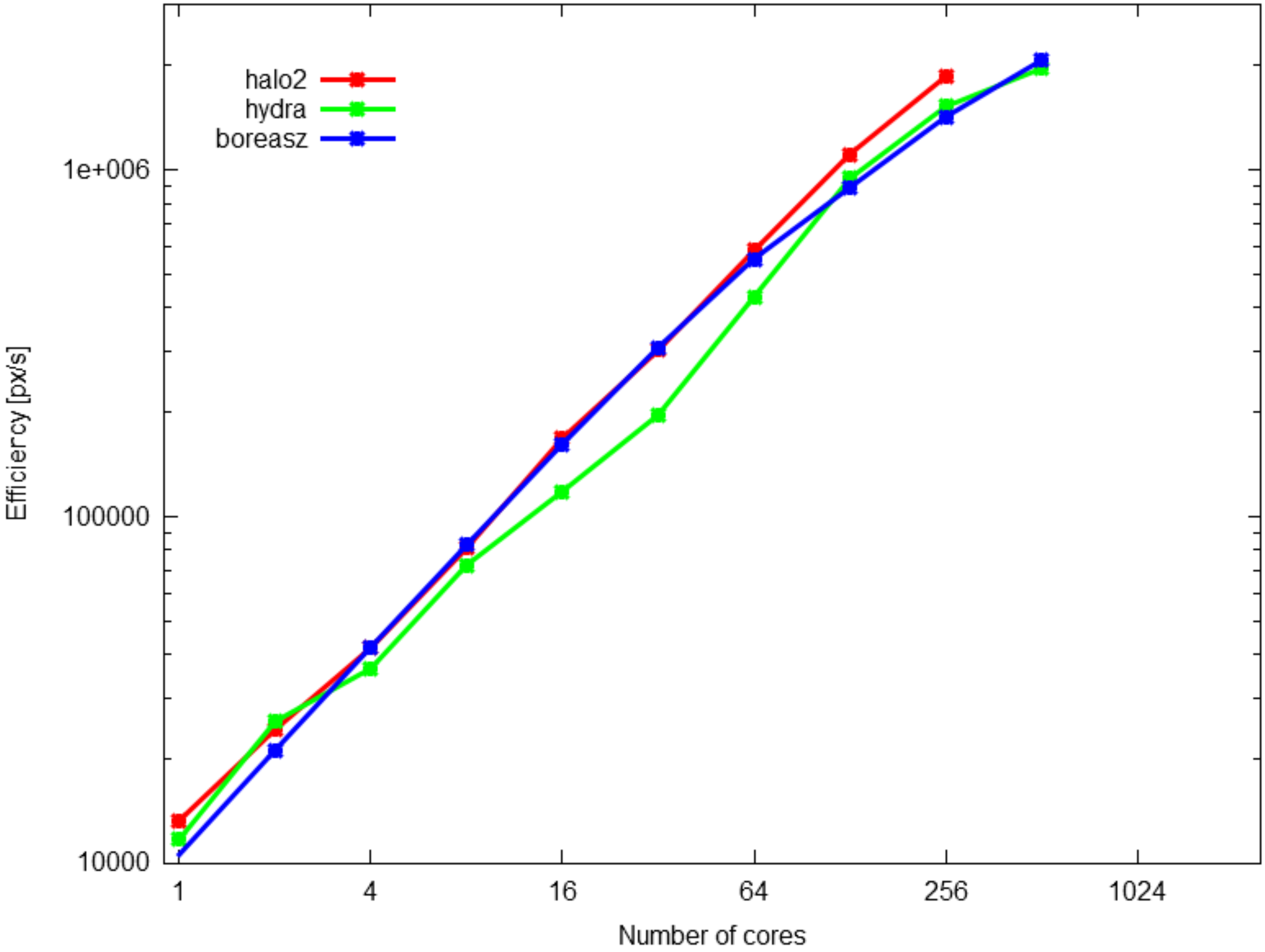
PCJ performance – Random Access



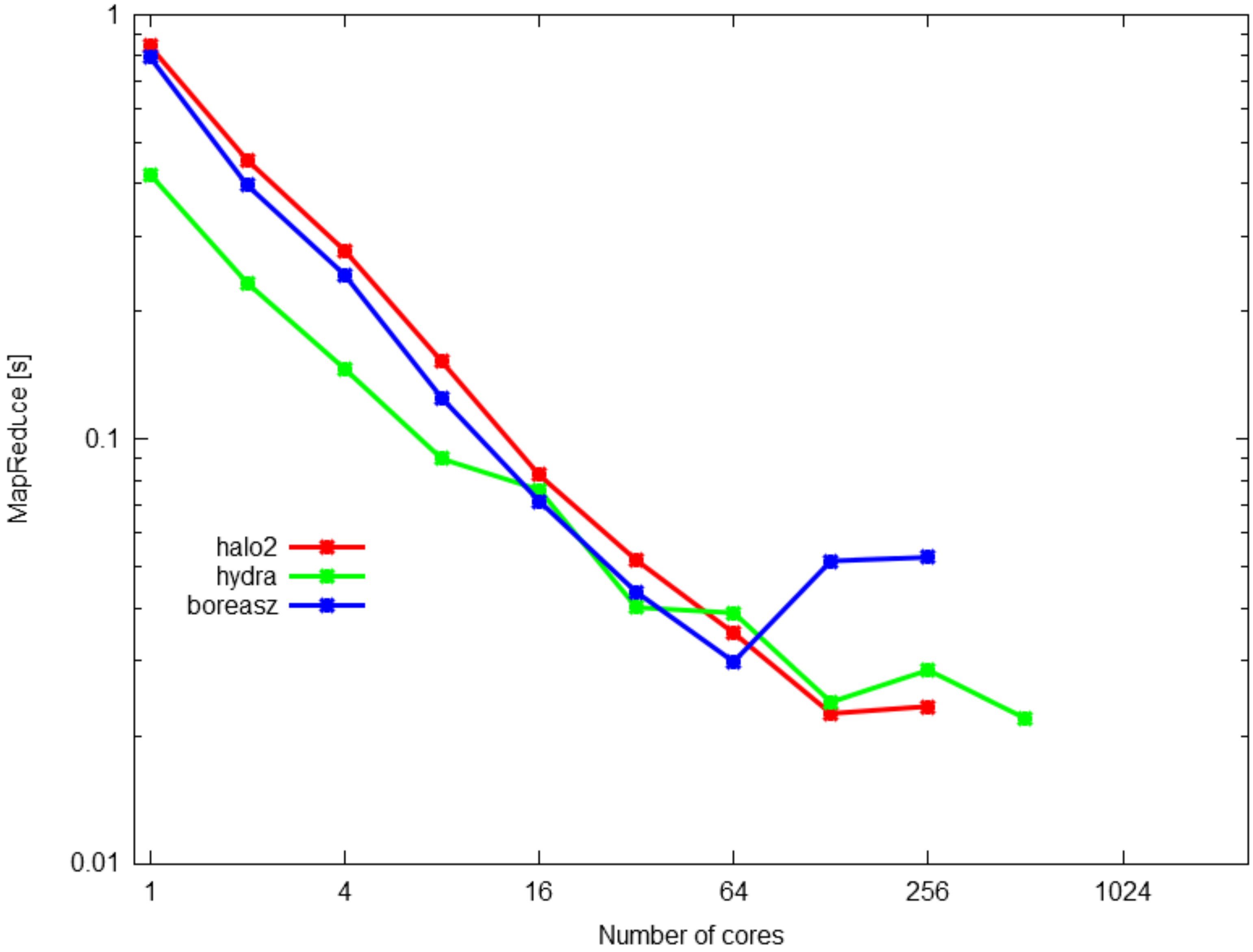
PCJ performance – Global FFT



PCJ performance – Raytracer



PCJ performance – MapReduce



PCJ for HPC and BigData

- For single node PCJ performance is competitive compare to Java 8 parallel streams
- PCJ performance is competitive compare to standard solutions based on MPI
- PCJ runs on multiple nodes (multiple JVM)
- PCJ has very good scalability and has been run on 10k cores
- PCJ can be used to parallelize data analysis codes written in Java

HPDCJ Project (CHIST-ERA)

Heterogenous parallel and distributed computing with Java

- Partners
  - ICM University of Warsaw (Warsaw, Poland)
  - IBM Research Lab (Zurich, Switzerland)
  - Queen's University of Belfast (Belfast, UK)
  - Bilkent Üniversitesi (Ankara, Turkey)
- Focus
  - ease of use and programmability of Java for distributed heterogeneous computing
  - heterogeneous systems including GPU and mobile devices
  - dependability and resilience by adding fault tolerance mechanisms
  - key applications including data-intensive Big Data applications
- 1st October 2014 – 31st September 2017
- **pcj.icm.edu.pl/hpdcj**

pcj.icm.edu.pl

Piotr Bała (ICM University of Warsaw)
Marek Nowicki (WMiI UMK)
Łukasz Górski (WMiI UMK)

bala@icm.edu.pl